MODELING PARTITIONED AND REPLICATED DATABASES USING AN EXTENDED CONCEPT OF GENERALIZATION*

Stanley Y. W. Su Amrish Kumar**

Database Systems Research and Development Center
University of Florida, Gainesville, Florida

ABSTRACT

Partitioning and replication of data are very important concepts in logical database design of both centralized and distributed databases. It is very important that the relationships among the vertical and/or horizontal partitions of a relation and the information about data replication be explicitly modeled and dynamically maintained so that their integrity and consistency can be automatically enforced by a DBMS or a distributed DBMS. In this paper, we present an extended generalization concept and show how it can be used to explicitly model various types of data partitions with or without replication. Storage operation rules for these partition types are defined for integrity control purposes and rules for modifying the schema at run time, when partitions are dynamically created or modified, are also presented. Integrity control can be enforced by following the modified model and constraints of these partitions.

1. INTRODUCTION

A database contains time-varying operational data of an enterprise and is defined by a schema. Partitioning in database design is the process of assigning a logical concept (relation) defined in the logical schema to several physical objects (files) in a stored database. For a distributed database the notion of partitioning is equivalent to the idea of fragmenting some relation and storing the fragments at different sites in the distributed database. Partitioning a database will thus involve the following decisions: 1) logical decisions, concerning the structure and composition of the fragments of a relation, and 2) distribution and allocation decisions concerning the placement of fragments at various sites.

The partitioning of a data file can be achieved in two ways. First, the columns (attributes) of a relation can be semantically categorized and partitioned to form several separate entity types in the conceptual design of a database. Each entity type is defined over a smaller number of attributes compared to the original relation. This method is referred to as VERTICAL PARTITIONING. Secondly, a relation can be subdivided into groups each of which contains tuples that satisfy a certain predicate. All groups have the same attributes of the original relation. This mode is referred to as HORIZONTAL PARTITIONING. In many cases, however, it is advantageous to partition a relation both horizontally and vertically, thus yielding blocks/ cells of tuples or subtuples from the original relation.

The motivation behind partitioning a database are fourfold:(1) Assuming that each site is an independent database that is part of a homogeneous or heterogeneous database, partitioning increases the locality of data at each site and allows, at the same time, access to the data from other sites. (2) Partitioning is used during the design of a database to improve the performance of transactions (faster access speed). Since the fragments consist of less/smaller records, fewer pages in secondary memory need to be accessed to process a transaction. Fragment allocation should maximize the amount of local transaction processing if the fragments match the requirements of transactions at a particular site. (3) Partitioning reduces the data transmission cost involved in moving data from one site to another. This cost may or may not be significant depending on how many sites contain the relevant data, the actual sizes of data involved in transmission, and the network structure and speed. (4) Finally, partitioning is one solution to the problem of storage limitation at network sites. In such cases, the cost and feasibility of storage expansion must be weighed against the added data communication costs.

If a database is partitioned and some data is replicated at multiple sites in a network system, it is important for the database management system to keep track of the relationships among the partitioned data files and the locations where the replicated data reside so that database integrity can be systematically enforced. A number of constraints are thus required to be associated with these partitions so that the distributed DBMS can dynamically monitor the partitioning and replication relationships of the fragmented data files and enforce consistency and integrity of the database. These constraints take the form of storage operation rules, which dictate what changes must be made to related partitions, if a change is attempted on some partition. The partitioning, replication, and distribution information along with the storage operation rules are not modeled explicitly in the present distributed database management systems, i.e. they are not included in the schema definition of a database. In order for a distributed DBMS to enforce the integrity of partitioned databases, they need to be defined by the DBA somewhere, if not in the schema. Since partitioning, replication and distribution are generally based on the semantic properties of the data and their usage statistics, we believe that they should be a part of the conceptual model of a distributed database.

*This research is supported by the Navy Manufacturing Technology Program through the National Bureau of Standards grant #60NAB4D0017.
**Current address:Honeywell Computer Sciences Center, 1000 Boone Ave. North, Golden Valley MN55427.

To appear: Proc. of COMPSAC '87, Tokyo, Japan, Oct. 1987.

A large body of work on Partitioning has been done on the selection and allocation of fragments in distributed databases, taking into account semantics of the data, user requirements and statistics, and minimization of access time [1 - 9]. In this paper, we do not concern ourselves with the above mentioned problems but propose a representation technique for partitioning information using an extended concept of generalization. This technique allows the representation of partition information as part of the global conceptual view of a distributed database.

The concept of **GENERALIZATION** was introduced by works of artificial intellegence (e.g. [12]). It was first used in database modeling by Smith and Smith [11]. This concept is a powerful tool for the expression and solution of a number of problems related to database modeling. The importance of generalization in database integration is emphasized in the MULTIBASE project [16,17] where it is used to resolve several structural and data inconsistencies that might exist within the different schemas to be integrated [18]. Properties of generalization hierarchies are examined in [10]. Generalization could also be used to resolve similar inconsistencies in the related but different problem of view integration [19]. In the context of database partitioning, we observe that the partitions are very closely related to each other as they are formed from the same relation, and contain information about the same entity type and can thus be treated as members of a more general class. By extending the concept of generalization to include set relationships as constraints, generalization can be used to model partitioned databases.

This paper is organized as follows: Section 2 describes the generalization construct of a Semantic Association Model (SAM*) and its set relationships. In section 3, the modeling of horizontal and vertical partitioned data with /without replication using the extended generalization is presented. Included in this section is a discussion of the storage operation rules, which must be applied to related partitions during update in order to maintain database consistency and integrity. Section 4 is a discussion on the maintaining of partition information at run time.

2. THE SAM* GENERALIZATION CONSTRUCT

In this section, we briefly describe the Semantic Association Model (SAM*) [20,21] and emphasize on the notion of generalization supported by it. Several constructs of SAM* will be used to model horizontal and vertical partitioned databases in section 3.

Data modeling in SAM* revolves around the notion of CONCEPTS(or OBJECTS) and ASSOCIATIONS. The model distinguishes two general types of concepts : ATOMIC and NON-ATOMIC. An atomic concept is a non-decomposable, observable physical object, abstract object, event or any data element that the database user regards as a fundamental information unit and whose meaning is assumed to be understood and thus need not be defined. An employee's age '25', the name 'John' are examples of atomic concepts. A non-atomic concept on the other hand, is a physical object, abstract object or event whose meaning is defined (described) in terms of other atomic and/or non-atomic concepts. For, example the concept of an employee can be described using the concepts name, age, address, and salary.

Atomic and/or non-atomic concepts can be grouped together to describe another non-atomic concept. This grouping is called an ASSOCIATION. Different types of associations can be distinguished based on the different structural properties,operational characteristics and semantic constraints that the user or DBA wants to associate with these groupings of concepts. If the user or DBA specifies that a concept is of a certain association type, a DBMS using SAM* would process the concept in accordance with the semantics associated with that type.

SAM* provides the user with seven such associations (modeling constructs). A detailed description of SAM* is out of the scope of this paper. We will, however, examine the generalization association in detail as it has been found useful for modeling partitioned databases with/without replication.

The notion of generalization was first used by Smith and Smith in database work. The generalization association in SAM* is defined in much the same way in that concepts can be grouped together based on their generic nature to form a more general concept. However a few enhancements have been made to the original concept to make it more useful in data modeling.

In SAM*, a generalization association is formed by grouping a number of generically related concept types which can themselves be defined by the same set or different sets of attributes. The key attributes of these component concept types, however, must have the same underlying domain; that is they must draw their values from the same set of data elements. The set of entities which are uniquely defined by the set of key attribute values of a component concept type may or may not overlap with that of another component concept type (the sets may or may not be exclusive). For example in figure 1, the two **AGGREGATION** nodes FOREIGN_PROJECT and DOMESTIC_PROJECT, representing two kinds of projects in a factory can be grouped together to form the more general concept of PROJECT.

In the graphic representation, the nodes represent concept types. Each node is labelled by its association type (G for Generalization, A for Aggregation and M for Membership). It is also named for user reference. The directed arcs in the graph represent attributes whose underlying domains are pointed to by the directed arcs. The crossed arcs represent the key attributes. The membership association (M nodes) define the domains P#, PNAME, AMT and DEPT respectively. Each domain contains a set of homogeneous data elements. The aggregation association (A nodes) define entity types by their attributes (arcs). The occurrences of an aggregation association are drawn from the cartesian product of the domain value sets.

As illustrated in figure 1, the set of attributes that defines the component concept type FOREIGN_PROJECT is different from that of the other component concept type DOMESTIC_PROJECT. Their key attributes are, however, defined over the same domain (the same membership association, P#). The G node PROJECT is the generalization of FOREIGN_PROJECT and DOMESTIC_PROJECT. Its occurrences can be formed by either taking the OUTERJOIN [13,14,15] of the occurrences of FOREIGN_PROJECT and DOMESTIC_PROJECT over their common key attribute or the union of their key values. The former representation is more suitable for presenting the generalized concept type to the user and the latter is a more condensed form suitable for internal machine representation.

The operation of outerjoin is an enhancement of the normal JOIN operator in relational algebra, introduced to preserve the information of unmatched tuples which is lost during a normal join operation. In taking the outerjoin, we preserve such information by appending certain additional tuples to the result of the normal join. There is one additional tuple for each unmatched tuple from each of the relations. It consists of a copy of that unmatched tuple, extended with null values in other attribute positions. Figure 2 illustrates an example of an Outer Equijoin. Several constraints that can be specified with the generalization association in order to make it more meaningful for data modeling. Among other applications, these constraints have been found useful for modeling CAD/CAM data and partitioned databases. The constraints can be viewed as relationships between the sets of key attribute values of two component concept types of the G node.

Figure 1 illustrates the constraint of Set Exclusion (SX). The two sets of entities represented by the occurrences associated with the aggregation nodes FOREIGN_PROJECT and DOMESTIC-_PROJECT are mutually exclusive. That is to say that no key value that represents a foreign project is allowed to represent a domestic project. If P#=1 is a foreign project, then there can be no domestic project with P#=1.

Figure 3 illustrates the constraint of Set Equality (SE), which specifies that any entity of GEOMETRIC_INFOR must also be an entity in MATERIAL_SPEC and LIFT_SPEC and vice versa. In the case of the non-key attributes that define the three aggregation nodes being different, the existence of one key value in one node (Design =357 in the occurrence of GEOMETRIC_INFOR) would require that the same key value be in an occurrence of MATERIAL_SPEC and also in an occurrence of LIFT_SPEC even though the non-key values of these occurrences may differ or may have null values.

Figure 4 shows the constraint Set-Subset (ST-SS) which specifies that a TOP_SECRET_PROJ is a PROJECT but a PROJECT may or may not be a TOP_ SECRET_PROJ. Again, we are considering the constraint in terms of the key value, that is, any key value which represents a TOP_SECRET_PROJ must be among the set of key values that represent PROJECT but the reverse may not be true.

In figure 5, we illustrate the SET INTERSECTION (SI) constraint in which the sets of keys which represent occurrences of the component concept types may overlap, that is, the intersection of the two sets may not be empty. Figure 5 models the fact that a MANAGER may or may not be a PROJECT_LEADER and vice versa.

It should be clear from the above examples, that the constraints associated with the generalization association are the set relationships between the sets of entities represented by the occurrences of the component concept types. If more than two sets are involved, the set relationships among them can be graphically represented by explicitly linking the labeled arcs between each pair of component concept types as illustrated in figure 3.

Although the examples seem to suggest that a generalization association can only involve concepts formed by the aggregation associations, this, in general is not true. It can be used to generalize concepts formed by all association types of SAM* (including itself). For example, a G node PART# can be formed over two M nodes FOREIGN_PART# and DOMESTIC_PART#.

We note here that even though a generalization can be defined over associations other than aggregation, the component concept types must be defined by the same association type. Even though it is theoretically possible to have a generalization whose components are defined by different association types, in practice it does not seem to be useful to generalize dissimilar concepts. The grouping of dissimilar objects is however meaningful in certain situations and is achieved using the COMPOSITION association of SAM*.

The so-called notion of Attribute Inheritance can also be modeled in SAM* as shown in figure 6. Here all the entity types forming the generic type have a common attribute. Instead of repeating the attribute WEIGHT in the definition of LAND-VEHICLE, WATER-VEHICLE, and AIR-VEHICLE, it is used as a component of the aggregation association VEHICLE_CHAR in which the generic type VEHICLE is the key attribute whose values are the set of all VEHICLE occurrences, and WEIGHT is the other attribute.

We note here that, as with the other association types, a Generic Hierarchy can be defined for the generalization association. For example, MAN_MADE_THING is a generic type over TRANSPORTATION_EQUIPMENT, which is a generic type over VECHICLE, etc.

We summarize the structural properties and constraints of the generalization association :

(1) The occurrences of a generic type are formed by either taking the outerjoin of its component concept types C1, C2, . . . Cn, or the union of the key attributes of C1, C2, . . . Cn. In case of the outerjoin representation, the attributes of the generic type is the union of the attributes of its component concept type.

(2) The generic relationship of concepts formed by any type of association can be explicitly modeled by generalization. These relationships and the associated constraints are treated as meta-data and are stored in the data dictionary.

(3) N-concepts can be grouped to form a generic type (N>1). These concepts can be defined by the same or different sets of component concept types but their key attributes must be defined over the same domain.

(4) The set relationship between each pair of the component concept types may be of type set-subset, set intersection, set exclusion or set equality. These set relationships are the integrity constraints associated with the generalization.

3. MODELING PARTITIONED DATABASES

Having discussed the needs and concept of horizontal and vertical partitioning of data and the notion of generalization, we now deal with the problem of modeling partitioned databases with /without replication.

A study of distributed database management systems and the kind of data that needs to be partitioned has led us to identify the following useful types of partitions :

(1) Horizontal Partitioning without Replication.
(2) Horizontal Partitioning with partial Replication.
(3) Horizontal Partitioning with complete Replication.
(4) Horizontal Partitioning with Subsetting.
(5) Vertical Partitioning with Replication.
(6) Vertical Partitioning without Replication.
(7)-(14) combination of (1)-(4) with each of (5), (6).

For each of the above cases, we show what the partition means in terms of the relational model and explain how it can be modeled using the SAM* generalization construct. We also list the insertion/deletion/update constraints on the partitioned data necessary to enforce database integrity and consistency.

We treat the primary key as just another attribute and allow it to be modified. For each of the partition cases the following general rules must hold at all times :

(1) **ENTITY INTEGRITY RULE** : The primary key value cannot be null in any of the relations (non-key attributes may or may not be null).
(2) **UNIQUE KEY CONSTRAINT** : The primary key value cannot be replicated in any of the relations.

3.1 Horizontal Partitioning without Replication

Figure 7(a) illustrates the case of horizontal partitioning without replication where a certain set of tuples from R forms R1 and another set forms R2. There is no overlap between the two sets of tuples. The situation can be modeled as shown in figure 7(b) where R1(A, B1, B2, B3, B4) and R2(A, B1, B2, B3, B4) are shown as aggregation nodes (The attribute A being the key in both the relations). The constraint SX on the G node ensures that there is no overlap of key values in these two sets of occurrences. The C nodes representing site1 and site2 indicate that R1 is present at site1 and R2 is present at site2. A C node represents the composition association in SAM* and groups similar or dissimilar concept types. It identifies a sub-database within a database.

CONSTRAINTS :-

INSERTION : Before inserting a tuple (A, B1, B2, B3, B4) in R1 or R2, verify that the key value (thus the tuple) does not exist in R2 or R1 respectively.

DELETION : No constraints.

UPDATE : Before modifying the key value in R1/R2 ensure that the key value does not exist in R2/R1.

3.2 Horizontal Partitioning with Partial Replication

Figure 8(a) illustrates the case of horizontal partitioning with partial replication. The key values of the tuples of R1 and R2 (A, B1, B2, B3, B4) are allowed to overlap. This fact is implied by the SI constraint on the G node (figure 8(b)). The relations R1 and R2 are represented by aggregation nodes and reside at site1 and site2 respectively, as shown by the C nodes.

CONSTRAINTS :-

INSERTION : Before inserting a tuple in R1 or R2, check if that value of A is present in R2 or R1. If the value exists in the other relation, then verify to ensure that the non-key attributes of both tuples have the same values. In the case that the value of A is not present in the other relation, check with the overlap condition specified to see whether it also should be inserted in the other relation.

DELETION : If the tuple to be deleted from any one of the relations is present in the other relation make the deletion in both relations.

UPDATE : Check with the overlap condition specified to see if the tuple that is being modified exists in the other relation. If it does, make the same modification to the other relation.

3.3 Horizontal Partitioning with Complete Replication

Figure 9(a) illustrates the case of horizontal partitioning with complete replication. The figure illustrates the simple case of R=R1=R2 but in the more general case we can have R1=R2=some subset of R. This situation is modeled in figure 9(b) where the SE constraint on the G node ensures that every occurrence of R1 is an occurrence of R2 and vice-versa. The A nodes represent the partitioned relations and the C nodes the sites.

CONSTRAINTS :-

INSERTION : Every tuple inserted in R1/R2 must be inserted in R2/R1.

DELETION : Every tuple deleted from R1/R2 must be deleted from R2/R1.

UPDATE : For every tuple modified in R1/R2, the same changes must be made to the corresponding tuple (same key value) in R2/R1.

3.4 Horizontal Partitioning with Subsetting

Figure 10(a) illustrates the case of horizontal partitioning with subsetting. The set of tuples that constitute R2 is a subset of the set of tuples that constitute R1. The figure illustrates the special case of R=R1 but in general this may not be true. The above situation can be modeled as shown in figure 10(b) where the ST-SS constraint on the G node ensures that every key occurrence of R2 is a key occurrence of R1, thus every tuple of R2 is a tuple of R1. The A nodes, as usual, represent the relations R1 and R2 and the C nodes the Sites.

CONSTRAINTS:-

INSERTION : Every tuple inserted in R2 must be inserted in R1 and every tuple inserted in R1 must be checked against the subsetting criteria specified to see if it should also be inserted in R2.

DELETION : Every tuple deleted from R2 must be deleted from R1 and every tuple deleted from R1 must be checked to see if it exists in R2. If it does, then it must be deleted from R2 also.

UPDATE : Every tuple modified in R2 must be accompanied by the same modification to the corresponding tuple in R1. For every tuple modified in R1, check if that tuple exists in R2 and make the same modification to the tuple if it does.

3.5 Vertical Partitioning without Replication

Figure 11(a) illustrates the case of vertical partitioning without replication. As mentioned earlier, a vertical partitioning must involve the replication of the key attribute. When we talk of replication, we are considering only the non-key attributes. The partitioning of R(A, B1, B2, B3, B4) into R1 (A, B1, B2) and R2(A, B3, B4) can be modeled as shown in figure 11(b). The Set Equality constraint ensures that every occurrence of R1 has a corresponding occurrence in R2 which is what is required since we are splitting the relation R vertically. R1, R2 represented as aggregation nodes have non-key attributes (B1, B2) and (B3, B4) respectively.
CONSTRAINTS:-
INSERTION : For every tuple (A, B1, B2) inserted in R1 a corresponding tuple (A, B3, B4) must be inserted in R2 and vice-versa. The attributes B1, B2, B3, B4 may have null values.
DELETION : For every tuple (A, B1, B2) deleted from R1 the corresponding tuple (A, B3, B4) must be deleted from R2 and vice-versa.
UPDATE : The non-key attributes in both relations may be modified with no constraint on the other relation unless there is an inter-occurrence constraint associated with the updated attribute values. Any change made to the primary key value, A, must however be reflected in the other relation.

3.6 Vertical Partitioning with Replication

Figure 12(a) illustrates the partitioning of R(A, B1, B2, B3, B4) into R1(A, B1, B2, B3) and R2(A, B3, B4). In the above case the non-key attribute B3 is replicated in both relations R1 and R2. The situation can be modeled as shown in figure 12(b), where as usual the A nodes represent the relations and the C nodes the sites. The SE constraint ensures that for any half of the tuple (of R) present in R1 the other half (of the same tuple of R) exists in R2.
CONSTRAINTS:-
INSERTION : Every tuple inserted in R1 must have a corresponding insertion in R2 where the A and B3 values must be the same and B1, B2, B3, B4 may be null.
DELETION : For every tuple (A, B1, B2, B3) deleted from R1, the corresponding tuple (A, B3, B4) must be deleted from R2.
UPDATE : All non-key, non-replicated attributes of R1 and R2 may be changed independently of the other relation but changes made to A or B3 must be carried over to the other relation.

In combination cases, horizontal and vertical partitioning are applied simultaneously to yield blocks or cells of data from the original relation. In such cases the storage operation constraints can be derived from the storage operation constraints associated with the particular type of horizontal and vertical partitions applied. In order to illustrate this, we discuss one example of combination partitioning. The other cases are similar in nature.

HORIZONTAL PARTITIONING WITH SUBSETTING AND VERTICAL PARTITIONING WITH REPLICATION

Figure 13(a) illustrates the extension of a relation R. R1 and R2 are combination partitions of the original relation where the horizontal partition associated with R2 is a subset of the horizontal partition associated with R1 (all key occurrences present in R2 are present in R1). The vertical partitions associated with R1 and R2 contain the replicated attribute B3. The extensions of R1 and R2 are shown in figures 13(b) and 13(c). Figure 13(d) illustrates the SAM* representation of the above situation.
CONSTRAINTS :-
INSERTION : If a tuple (an, bn3, bn4) is inserted in R2, the corresponding tuple (an, bn1, bn2, bn3) must be inserted in R1. If (am, bm1, bm2, bm3) is inserted in R2, the subset criteria must be checked to see if the insertion (am, bm3, bm4) in R2 is necessary.
DELETION : If a tuple is deleted from R1, the corresponding tuple must be deleted from R2. If a tuple is deleted from R2, the semantics of the subsetting must be checked to see if the corresponding tuple must be deleted from R1.
UPDATE : If some attribute(s) is modified in R1/R2 and if that tuple and attribute exists in R2/R1, the same modifications must be made to the corresponding tuple in R2/R1.

4. MAINTAINING PARTITION INFORMATION

In this section, the dynamic nature of the partition information, which is defined by a SAM* schema and graphically represented by a semantic network will be discussed. So far we have made no mention of how the semantic network can be modified to incorporate changes the user may make to the existing partition information. In this section, we discuss how the network should be modified when a change is requested.

The user may want to do any of the following three operations:
(1) **Create a new partition.** The user may want to further partition R1 into R1' and R1" (figure 14), or create another partition R3 from R (figure 15).
(2) **Delete an existing partition.** The user may want to remove partitions R1 or R2 from the semantic network.
(3) **Modify existing partitions.** The user may want to shift attribute B1 from R1 to R2.

For each of the above cases, the changes to be made to the network to realize the user request will be discussed. Before that, however, the problem of how the user should communicate his request to the system is considered first. One solution is to allow the user to issue explicit commands to the system, naming the partitions affected and stating what he/she would like done. This would mean that the user be well versed with the semantic network representing the partitions. We feel that this goes against the notion of user-friendliness and the 'high-level' notion of a DBMS. The task should then be handled by the system for which we assume the existence of a powerful query language, which allows the system to infer the desired information from the query issued by the user. As an example, we consider the following query in an INGRES like language :

```
RETRIEVE * FROM employee INTO female AT site6
        WHERE sex='f';
```

The given query would enable the system to create a new partition FEMALE which contains only those tuples from the EMPLOYEE relation where sex is 'f'. The created partition is stored at site6, and the relationship between the two partitions is of type set-subset.

If the system now received another query :

```
RETRIEVE * FROM employee INTO male AT site2
        WHERE sex='m';
```

It would create another partition MALE to be stored at site2. Again the relationship between EMPLOYEE and MALE is of type set-subset. We have also to specify the relationship between MALE and FEMALE as in the network diagram the EMPLOYEE relation is represented as a G-node comprising of EMPLOYEE, MALE and FEMALE. Set relationships have to be specified for each pair. To achieve that, it is necessary to check the conditions by which these relations are formed.

Given any two partitions R1 and R2 of the relation R, there exist four possible ways in which they may be related :1) R1 and R2 are mutually exclusive, 2) R1/R2 is subset of R2/R1, 3) R1 and R2 are equal, and 4) R1 and R2 have some intersection of an unknown nature.

The nature of the relationship between R1 and R2 can be determined by examining the WHERE predicates of the queries that created the partitions. The examining of the Where clause can be handled most conveniently by using the Conjunctive Normal Forms (CNFs) of the above mentioned where predicates. We now present a discussion of the procedure to determine the relationship between any two partitions. For the purposes of this discussion, we assume that R1 and R2 are partitions of the relation R and the CNFs of the corresponding partitions are of the form CNF1 = (attr11 opr11 par11) and (attr12 opr12 par12) and . . . . . and CNF2 = (attr21 opr21 par21) and (attr22 opr22 par22) . . . .

(1) TESTING FOR MUTUAL EXCLUSION
Compare the first conjunct of CNF1 with every conjunct of CNF2 in turn. If there is no match between attr11 and any attribute of CNF2 the comparison process is repeated with the next conjunct of CNF2 until there is a match or all the conjuncts of CNF1 are exhausted without yielding a match.

In case no attribute of CNF1 matches any attribute of CNF2, R1 and R2 may or may not share an intersecting domain and we may conclude that R1 and R2 fall under case 4.

In case some attribute in CNF1 (attr1n) matches an attribute in CNF2 (attr2n), this pair is examined further for determining whether their associated conditions are mutually exclusive. For example, if opr1n (corresponding to attr1n) is '>=' and opr2m (corresponding to attr2m) is '<' and par1n = par2m = constant then R1 and R2 are mutually exclusive. In general we can pass the matching pair through rules of the form IF opr1 = ' ' and opr2 = ' ' and par1n OP par2m THEN the partitions are mutually exclusive, where OP represents a boolean operator.

In a case where there is more than one attribute matching, then any one of the pairs can render the partitions mutually exclusive since we are using the conjunctive normal form. If we cannot show the partitions to be mutually exclusive, we proceed to check if they form a set-subset relationship.

(2) TESTING FOR SET-SUBSET (R1 < R2)
If no attribute value in CNF1 matches any attribute value in CNF2, then we cannot conclude that R1 is a subset of R2 and we conclude case 4. If there exists at least one matched attribute, found in comparing CNF1 and CNF2 such that the CNF2 conjunct spans over the CNF1 conjunct (this can be determined by passing the matching conjuncts through rules of the type discussed in case (1)), then R1 can be concluded to be a subset of R2 provided that no conjunct of CNF1 spans over a conjunct of CNF2 and there are no additional conjuncts in CNF2 (no terms restricting the values of an attribute that is not present in CNF1).

(3) TESTING FOR SET-EQUALITY
If R1 is determined to be a subset of R2 in 2. then the possibility of set-equality between R1 and R2 must be verified. This is done by testing if R2 is a subset of R1 using the methodology outlined in 2. If R1 < R2 and R2 < R1, then we can conclude that R1 = R2.

(4) EXISTENCE OF SET INTERSECTION
This option is relevant when all the above tests fail to establish some definite relationship between R1 and R2. Here, we assume that R1 and R2 intersect in an undetermined fashion. This case thus serves as a default case and caters to all partitions which cannot be explicitly classified.

We now return to the problem of modifying the existing network, once a user request has been received and identified. As mentioned earlier, the system can receive three classes of requests. We discuss each one in turn through examples.

(A) CREATE A NEW PARTITION
Here we identify two types of requests :
(a) Partition a relation that is itself a partition of another relation. For example, partition R1 into R1' and R1" (figure 14).
(b) Create another partition from a relation that is already partitioned. For example create R3 from R in addition to R1 and R2 (figure 15).

For requests of type (a), we need to make the following modifications to the semantic network :
(a1) Convert the node representing the relation to be partitioned (R1) to a G node and remove all attribute pointers from the node.
(a2) Create as many A nodes as the number of new partitions desired. These nodes are the component concept types of the G node in (a1).
(a3) Create as many C nodes as the number of new (non-existant) sites required and connect them to the appropriate A nodes which represent the partitions being stored at these sites.
(4a) Create M nodes (may or may not be duplicated) for the attributes that comprise the resulting partitions. These attributes initially formed relation R1 and the resulting semantic network must be consistent with this notion. If some attribute of R1 had a relationship with another entity outside of the part of the network being considered, the relationship should

be maintained even after the modification.
(a5) Specify set relationships between each pair of A nodes created in (a2).

The suggested modifications transforms the semantic network into the one shown in figure 14. However, it should be noted that the relation R1 as shown in the figure is present only in the logical view of the database and does not exist physically. Therefore we can only make this transformation if the union of the set of tuples comprising R1' and R1" is the same as the set of tuples comprising R1. If this is not true, we will loose some information in making this transformation and need to physically store R1. In terms of the network it will mean creating another A node (R1) under the G node in addition to the ones in (a2) with a set-subset relationship with all the other nodes.

It is mentioned here that changing the A node representing R1 to a G node as suggested in (a1) is insignificant with regard to any previous relationships involving R1. The A node representing R1 before modification denoted a set of tuples constituting the relation R1. The same meaning is retained by changing it to a G node which represents an outerjoin of the constituent partitions.

For requests of type (b), we suggest the following modifications to the existing semantic network :
(b1) Create another A node representing R3 as another component concept type of the G node representing R.
(b2) Specify set relationships between the created node in (b1) and all other existing A nodes.
(b3) Create another C node, if required, to denote the site at which the new partition will be stored.
(b4) Show the attributes of the created partition R3 by pointers to the desired attributes.

The result of the transformation will be the semantic network shown in figure 15.

(B) DELETE AN EXISTING PARTITION

To delete the partition R1 from the database, we propose the following modifications :
(a) Remove the A node representing the partition R1.
(b) Remove all pointers to that node.
(c) Remove all pointers from that node.
(d) Remove all nodes and the pointers to and from the nodes that are existentially dependent on the node removed in (a).

The above transformation yields the network diagram as shown in figure 16.

We note from the figure that the G node present no longer serves any useful purpose and should be deleted from the semantic network. The removal of the G node, however, may have additional triggering effects since it may be involved in other associations and the consequences of its removal must be propogated throughout the network.

Let us assume that after the transformation applied earlier, the network is as shown in figure 17(a). We now discuss the consequences of deleting the G node, in view of its relationships with nodes X1 through X4. It should be mentioned here that this discussion is applicable only to G nodes as in figure 17(a) which have only one constituent type and are involved in relationships with other nodes.

For the A and I nodes (I node in SAM* models the relationship between/among independent concept types), the removal of the partition R1 translates to the removal of all tuples from the set whose key value matches any of the key values originally present in R1 but not in R2. This modification is necessitated due to the fact that R1 is now non-existant and it's occurrences should be deleted from wherever they existed. The G node can now be removed and the nodes X1, X2 can be connected directly to the A node representing R2.

For the C node, it suffices to just remove the G node and connect it directly to the A node representing R2 since the C node is a collection of the occurrences of it's component concept types.

The above mentioned procedure, to account for the A, I and C nodes can be applied recursively to the G node (X4) provided it meets the requirements of having only one component concept type and being involved in relationships with other nodes. The resulting transformation is shown in figure 17(b).

(C) MODIFYING A PARTITION

The operation of modifying a partition (add B1 to R2) involves the shifting of attributes from one partition to another and may not be as useful as the previous two operations as it involves only a restructuring of the network and no addition or deletion of information. It can be realized by a change of pointers or depending on the request, a duplication of an existing node.

All the proposed modification steps should be treated as one operation which is valid only on the completion of all the steps. Partial completion may lead to serious problems, which may be difficult to retract. The creation of each new partition must also be accompanied by the storage of the partition criteria in the data dictionary. This would enable the system to maintain consistency in the database as tuples are inserted and will also be useful in defining set relationships should another partition be created at run time.

5. CONCLUSION

A modeling construct and some techniques for modeling and processing partitioned and replicated databases have been presented. The generalization association of SAM* and its associated set constraints are used to explicitly model fourteen meaningful combinations of partitioning and replication of data. Storage operation rules for forcing the integrity of data partitions have been presented. Strategies and rules for handling dynamic creation and modification of partitions using schema modification have also been described. The model, rules and techniques presented in this paper are useful for logical database design of both centralized and distributed databases.

Our future work in this area includes :(1) developing a methodology to determine an optimum fragmentation scheme based on user/transaction requirements.(2) investigating strategies for an optimum Fragment-Site allocation scheme in a heterogeneous environment.(3) developing a query modification algorithm to retrieve/update data from fragmented relations when a query makes reference to the original relation.

REFERENCES

(1) S.Ceri, M.Negri and G.Pellagati, Horizontal Partitioning in Database Design, ACM-SIGMOD, 1982.
(2) S.Ceri, S.B.Navathe and G.Weiderhold, Distribution Design of Logical Database Schemas, IEEE-TSE, SE-9:4, 1983.
(3) S.B.Navathe, S.Ceri, G.Weiderhold and J.Dou, Vertical Partitioning Algorithms for Database Design, ACM Transactions on Database Systems, Vol. 9, no.4, Dec. 1984, pp. 680-710.
(4) M.Hammer and B.Niamir, A Heuristic Approach to Attribute Partitioning, ACM-SIGMOD, 1979.
(5) M.J.Eisner and D.G.Severance, Mathematical Techniques for Efficient Record Segmentation in Large Shared Databases, Journal of the ACM, 23:4, 1976.
(6) W.W.Chu, Optimal File Allocation in a Multiple Computer System, IEEE-TC, C-18:10, 1969.
(7) H.L.Morgan and J.D.Levin, Optimal Program and Data Locations in Computer Networks, Communications of the ACM, 20:5, 1977.
(8) C.V.Ramamoorthy and B.W.Wah, "The Placement of Relations in a Distributed Relational Database," Proc. First Int. Conf. on Distributed Computing Systems, 1979.
(9) P.P.S.Chen and J.Akoka, "Optimal Design of Distributed Information Systems," IEEE-TC, C-29:12, 1980.
(10) R.M.Lee and R.Gerritsen, Extending semantics for generalization hierarchies, in Proceedings of ACM/SIGMOD International Conference on Management of Data, Austin, Tex., 1978.
(11) J.M.Smith and D.C.P.Smith, Database abstractions: Aggregation and Generalization, ACM Trans. on Database Systems 2(2): 105-333 (June 1977).
(12) Quillian, M.R., Semantic memory. In Semantic Information Processing, M.I.T. Press, Cambridge, Mass., 1968, pp.227-268.
(13) Date,C. "The Outer Join," in Proc. 2nd Intl. Conf. on Databases, Cambridge, England, September 1983, pp. 76-106.
(14) Dayal, U. "Processing Queries Over Generalization Hierarchies in a Multibase System," in Proc. 9th Conf. Very Large Data bases, August 1983, Milan, Italy, pp. 342-353.
(15) Reiner,D., and Rosenthal, A. "Extending the Algebraic Framework of Query Processing to Handle Outerjoins," in 10th Intl. Conf. Very Large Data Bases, Singapore, August 1984, pp. 112-120.
(16) Smith,J.M., P.A.Bernstein, U.Dayal, N.Goodman, T.Landers, K.W.T.Lin and E.Wong. "Multibase-Integrating heterogeneous distributed database systems". Proc. AFIPS NCC 1981. pp.487-499.
(17) Landers,T.,and Rosenberg, R. "An Overview of Multibase," in Proc. 2nd Intl. Symposium on Distributed Data Bases, H.J.Schneider (ed.), September 1982, Berlin, F.R.G., pp. 153-184.
(18) Dayal,U., and Hwang, H. "View Definition and Generalization for Database Integration in Multibase," IEEE Trans. on Software Engineering, Volume SE-10, Number 6, November 1984, pp. 628-644.
(19) Mannino, M., and Effelsberg, W. "A Methodology For Global Schema Design," University of Florida, C.I.S. Dept. Technical Report No. TR-84-1, September 1984.
(20) Su, S.Y.W., "SAM* : A Semantic Association Model for Corporate and Scientific-Statistical Databases," Information Sciences, 29, 1983, pp. 151-199.
(21) Su, S.Y.W., "Modeling Integrated Manufacturing Data Using SAM*," Proc. of Data Base Systems for Office, Engineering and Science, Karlsruhe, West Germany, March 1985, also IEEE COMPUTER, vol.19 no. 1, Jan 1986, pp. 34-49.

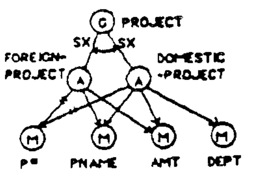

FIGURE 1: SET EXCLUSION CONSTRAINT

SUPPLIER :

S#	SCITY
S1	LONDON
S2	PARIS
S3	NEW YORK

PARTS :

P#	PCITY
P1	LONDON
P2	NEW YORK
P3	ATHENS

OUTER EQUIJOIN OF SUPPLIER AND PARTS :

S#	SCITY	P#	PCITY
S1	LONDON	P1	LONDON
S2	PARIS	NULL	NULL
S3	NEW YORK	P2	NEW YORK
NULL	NULL	P3	ATHENS

FIGURE 2: AN EXAMPLE OF OUTER EQUIJOIN

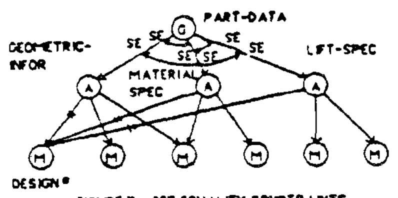

FIGURE 3 SET EQUALITY CONSTRAINTS

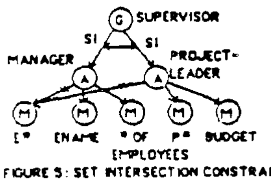

FIGURE 5: SET INTERSECTION CONSTRAINT

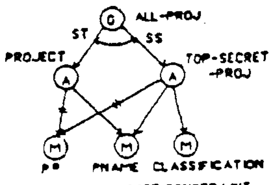

FIGURE 4 : SET SUBSET CONSTRAINT

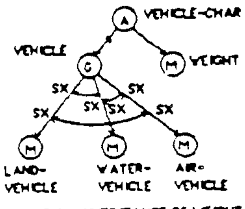

FIGURE 6 INHERITANCE OF WEIGHT

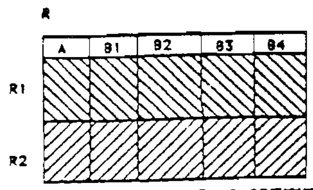

FIGURE 7(a) : HORIZONTAL PARTITIONS WITHOUT REPLICATION

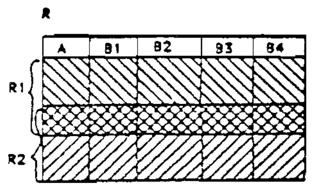

FIGURE 8(a) : HORIZONTAL PARTITIONS WITH PARTIAL REPLICATION

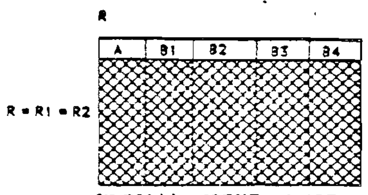

FIGURE 9(a) : HORIZONTAL PARTITIONS WITH COMPLETE REPLICATION

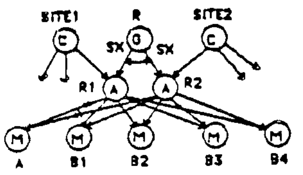

FIGURE 7(b) : THE MODEL FOR PARTITIONS R1 & R2

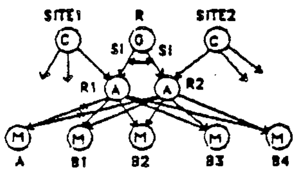

FIGURE 8(b) : THE MODEL FOR PARTITIONS R1 & R2

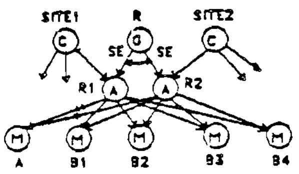

FIGURE 9(b) : THE MODEL FOR PARTITIONS R1 & R2

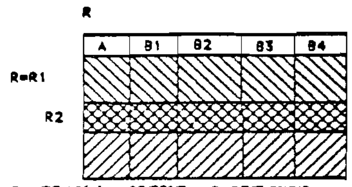

FIGURE 10(a) : HORIZONTAL PARTITIONING WITH SUBSETTING

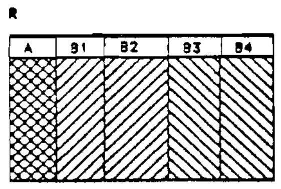

FIGURE 11(a) : VERTICAL PARTITIONING WITHOUT REPLICATION

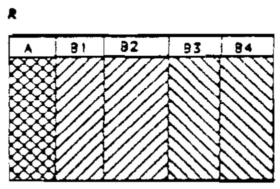

FIGURE 12(a) : VERTICAL PARTITIONING WITH REPLICATION

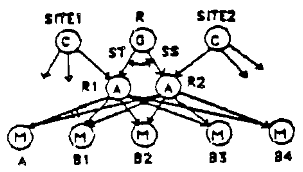

FIGURE 10(b) : THE MODEL FOR PARTITIONS R1 & R2

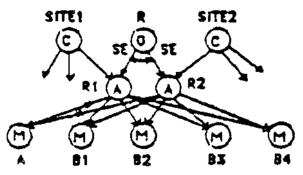

FIGURE 11(b) : THE MODEL FOR PARTITIONS R1 & R2

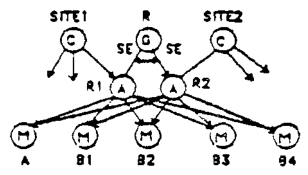

FIGURE 12(b) : THE MODEL FOR PARTITIONS R1 & R2

R

A	B1	B2	B3	B4
a1	b11	b12	b13	b14
a2	b21	b22	b23	b24
a3	b31	b32	b33	b34
a4	b41	b42	b43	b44

FIGURE 13(a)

R2

A	B3	B4
a2	b23	b24
a3	b33	b34

FIGURE 13(c)

R1

A	B1	B2	B3
a1	b11	b12	b13
a2	b21	b22	b23
a3	b31	b32	b33
a4	b41	b42	b43

FIGURE 13(b)

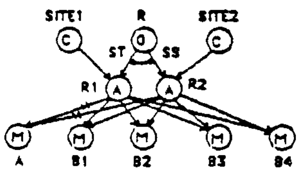

FIGURE 13(d) HORIZONTAL PARTITIONING WITH SUBSETTING AND VERTICAL PARTITIONING WITH REPLICATION

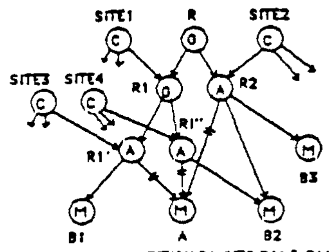

FIGURE 14 : PARTITION R1 INTO R1' & R1''

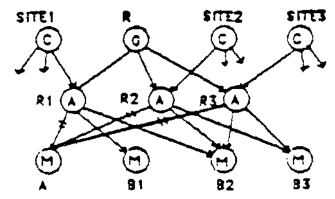

FIGURE 15 : CREATING R3 FROM R

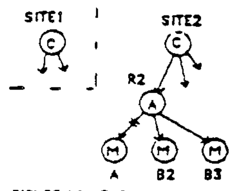

FIGURE 16 : THE DIAGRAM AFTER DELETING R1

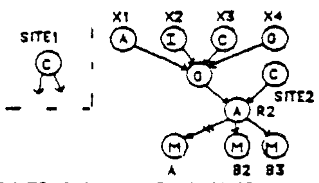

FIGURE 17(a) : THE NETWORK DIAGRAM AFTER TRANSFORMATION

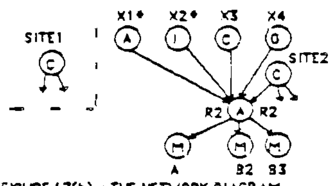

FIGURE 17(b) : THE NETWORK DIAGRAM AFTER MODIFICATION
* occurrences modified